CHAPTER 47

Introducing Dynamic HTML and the Document Object Model

In This Chapter

What is Dynamic HTML?

The Document Object Model (DOM)

Implementing the DOM

DOM properties

Once upon a time, if you wanted to interact with the user of a Web page, there were two options. First, you could stand over their shoulder and talk to them while they were browsing. This was very personal, but difficult if more than one person browsed your page at a time, or if the person happened to be in a faraway place such as Turkmenistan or Argentina.

The other option was to run your own Web server, write your own server-side scripts in a language such as Perl, and depend on the user having a reasonably fast connection to handle the network traffic required to handle the updates to the user's page. Because each page was essentially static, it required the server to send pages that prompted the user's browser to ask for a new page, which the server would generate and send back to the user.

Although the second method required far less travel than the first, it was still inaccessible for many people who did not have the programming knowledge required to work with server-side scripting, or were short of cash to hire someone who did. Then, Netscape introduced JavaScript and made it possible to add scripting directly to a Web page, which ran

directly on the user's browser without any additional help from the server. A new era was born. In this book, a new section begins.

This section is an introduction to Dynamic HTML (DHTML) — one of the most interesting and rewarding ways to manipulate a Web page. DHTML provides exciting and creative ways for developers and HTML authors to bring their pages to life. While this section is not an all-encompassing tutorial on DHTML, it gives you a taste of what's possible with a little work and imagination.

Lots of information is available from the leading browser companies. Check out the developer sites at Netscape (`http://developer.netscape.com`) and Microsoft (`http://msdn.microsoft.com`).

What Is Dynamic HTML?

Like an interactive page, Dynamic HTML can be a bit of a moving target. It's a technology and idea that's evolved since Netscape first introduced JavaScript in Navigator 2. But we all need a starting place, so let's break the term apart.

First, there's *dynamic*. In DHTML, *dynamic* means that the page is subject to change at any time. More specifically, this change does not require any interaction with a server — everything happens on the client side (the browser), so it's quick and doesn't depend on a fast connection or an optimized server.

Then there's HTML — this book is dedicated to building pages with HTML, so you've got a pretty good idea what it's all about. At the heart of HTML is the ability to define a page's structure so that a browser knows how to interpret the page for the user. Traditional HTML is static — you load the page and it's done. It just sits there like a good dog ready for you to read or move on to another page.

Putting the two together means we have a method of manipulating the structure of a Web page without getting the server involved beyond loading the initial page. What does this mean? This means suddenly you can do things on your Web page you couldn't do before, such as

- ✦ Redraw a page without reloading
- ✦ Move images and text around the page without using a movie or animation (see Figure 47-1)

- ✦ Create dynamic forms that automatically ask the next question based on the answers to the previous questions
- ✦ Create a dynamic table of contents (see Figure 47-2)
- ✦ Show and hide levels of detail based on visitor actions without reloading the page
- ✦ Change the color of text or an image as the cursor moves over it

Netscape 6 - What's New - Netscape
File Edit View Search Go Bookmarks Tasks Help
Bookmarks Home My Netscape Net2Phone Customize...
http://home.netscape.com/browsers/6/su_setup.html
SEARCH
My Sidebar customize...
CNN.com
What's Related
Search
Buddy List
Stocks
News
TOP STORIES (April 27, 9:58 PM)
U.S. Finally to Honor Powers, U2 Spy Plane Pilot
Reuters Top News Summary
Reno Defends Elian Raid, Court Rebuffs Relatives
SPORTS (April 27, 9:58 PM)
Spain, Australia, France Among Winners at Fed Cup
Reuters Sports News Summary
Search from anywhere, anytime
type a word here, then click Search
What's New
My Sidebar is yours
Click "customize" to choose from hundreds of tabs
Easy to open and close
Streamlined Interface
which you can customize with themes (coming soon)
Small and Speedy
for faster download and browsing
Same-time Messaging
email and instant messaging combined to keep you connected
Netscape
The Best Standards Support
for the most dynamic and robust web sites and applications
Learn More
Start Using Now
Convenient Services
always available and easy to find
Document: Done (7.74 secs)
Build ID: 2000033112
Channels Tools Business Free Time Shopping Go To Window
Start
Netscape 6 - What's...
9:24 PM

Figure 47-1: Words gyrate and circle on this Netscape page thanks to scripting harnessing the power of the DOM. This page uses layers to move text without using an animated GIF or other bulky graphic files.

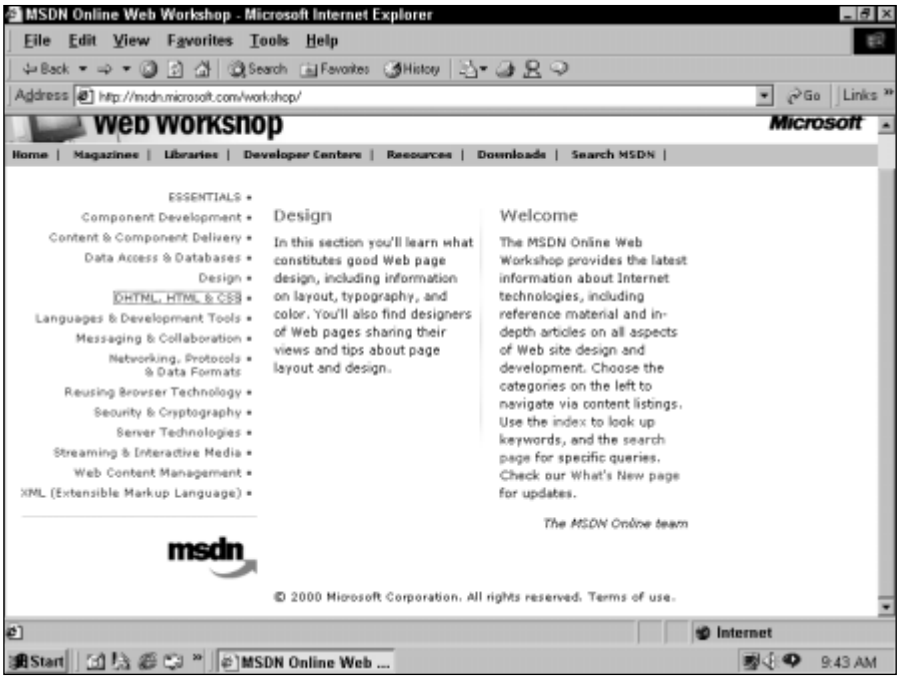

MSDN Online Web Workshop - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://msdn.microsoft.com/workshop/
Web Workshop
Microsoft
Home | Magazines | Libraries | Developer Centers | Resources | Downloads | Search MSDN |
ESSENTIALS
Component Development
Content & Component Delivery
Data Access & Databases
Design
DHTML, HTML & CSS
Languages & Development Tools
Messaging & Collaboration
Networking, Protocols & Data Formats
Reusing Browser Technology
Security & Cryptography
Server Technologies
Streaming & Interactive Media
Web Content Management
XML (Extensible Markup Language)
Design
In this section you'll learn what constitutes good Web page design, including information on layout, typography, and color. You'll also find designers of Web pages sharing their views and tips about page layout and design.
Welcome
The MSDN Online Web Workshop provides the latest information about Internet technologies, including reference material and in-depth articles on all aspects of Web site design and development. Choose the categories on the left to navigate via content listings. Use the index to look up keywords, and the search page for specific queries. Check our What's New page for updates.
The MSDN Online team
msdn
© 2000 Microsoft Corporation. All rights reserved. Terms of use.
Internet
Start
MSDN Online Web ...
9:43 AM

Figure 47-2: Menus appear and disappear as the mouse hovers over the words, which act as a table of contents on the left-hand side of this Microsoft page. This page uses the DOM to detect mouse position and display hidden layers.

In DHTML, the instructions for all of these features become a part of your Web page. All of the instructions are executed without the page being reloaded from a server. They go a long way toward converting a Web page into a true application.

However, none of the exciting things you can do with your page can be done without learning a scripting language. Because JavaScript is the most widely implemented and the most widely used, this book uses JavaScript for its examples. JavaScript is covered in more detail in the next chapter, although you'll begin to see how it looks in this chapter.

For lots of examples and in-depth knowledge of JavaScript, be sure to check out the *JavaScript Bible* by Danny Goodman from IDG Books Worldwide, Inc.

The ability to address and change tags and their content utilizes the Document Object Model (DOM). As browsers have evolved, virtually every element on a Web page is now reflected in the DOM and is exposed to change or observation.

Foundation for Change: The Document Object Model

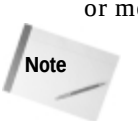
DHTML needs a way to look at the document. In static HTML, the page is loaded, interpreted, displayed, and that's it. It's simply a set of instructions on how the page should appear.

In DHTML, the elements are related to each other through the use of the Document Object Model (DOM). As each element is loaded, it is placed in a stack with similar elements with a note as to where it is in relation to other documents. In this way, it keeps track of everything on the page — in essence creating a mini-database reflecting the page's structure and content.

Without the DOM, the Web would never be more than a series of static magazine pages. Sure, some pages would look pretty sexy with plug-ins like Shockwave, animated GIF images, or pages created using a real-time database. But, essentially, everything you would ever see on a page would be the same, regardless of how you interact with the page. With the DOM, the Web becomes an application on your desktop. Using the DOM, every Web page can be a functional software application that responds to the actions of the user, including typing information in form fields, moving a mouse over words and images, and clicking.

The DOM makes all of this possible by creating a structured way to access elements in a document in an orderly fashion. It relates each element to its neighbors through the use of parent-child-sibling relationships. The DOM is not a new set of tags or attributes to learn, because the DOM's job is to work within existing document formats, including HTML and XML.

The DOM standard is maintained by the World Wide Web Consortium, or W3C (`www.w3c.org`), which works hard to create universal recommendations that aren't application- or vendor-specific (see Figure 47-3). The DOM as described by the W3C is language- and browser- independent. This is a step forward from the first implementation of a DOM, which was created and interpreted solely in context of Netscape Navigator 2.0, where the DOM was tied directly to JavaScript. The DOM created by the W3C lays the groundwork for how a document's properties are accessed, but not the structure or method.



All the information you would ever want on the DOM is located at the W3C's Web site (`www.w3c.org/DOM`). This includes history, current status and future plans for the DOM, and the current DOM recommendations and drafts.

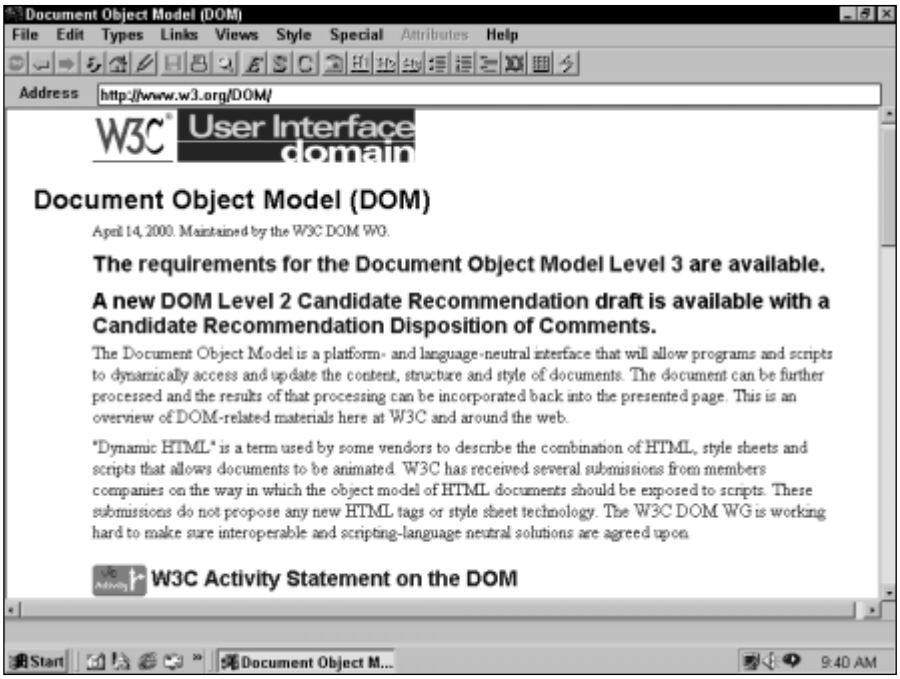

Document Object Model (DOM)
File Edit Types Links Views Style Special Attributes Help
Address http://www.w3.org/DOM/
W3C User Interface domain
Document Object Model (DOM)
April 14, 2000. Maintained by the W3C DOM WG.
The requirements for the Document Object Model Level 3 are available.
A new DOM Level 2 Candidate Recommendation draft is available with a Candidate Recommendation Disposition of Comments.
The Document Object Model is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents. The document can be further processed and the results of that processing can be incorporated back into the presented page. This is an overview of DOM-related materials here at W3C and around the web.
"Dynamic HTML" is a term used by some vendors to describe the combination of HTML, style sheets and scripts that allows documents to be animated. W3C has received several submissions from members companies on the way in which the object model of HTML documents should be exposed to scripts. These submissions do not propose any new HTML tags or style sheet technology. The W3C DOM WG is working hard to make sure interoperable and scripting-language neutral solutions are agreed upon.
W3C Activity Statement on the DOM
Start
Document Object M...
9:40 AM

Figure 47-3: The World Wide Web Consortium officially maintains the standards for the Document Object Model, and posts the most current revisions on its Web site.

How is the DOM implemented?

The DOM by itself doesn't do anything, but it does provide a way to address any object on your page. In previous chapters, each *thing* on your page was referred to as an *element*. Now, with the Document Object Model, suddenly everything becomes an *object*. What's the difference? Not much. An element is something you define for display only. In the DOM, the element becomes an object to act upon.

Typically, the browser creates a document-specific DOM from the elements on a Web page. A scripting language (usually JavaScript) is then used to interact with the user and environment to manipulate the page's appearance or behavior. In the DOM, every object can be referred to either by name or by its relationship to other objects on the page. You may have noticed that nearly every element in the HTML 4.01 specification has `name` as one of its attributes. You can use this attribute to assign a name that won't be of any use anywhere except in the script, using the DOM.

Every element in HTML has an associated object name in the DOM, which enables you to refer to objects by relationship. Most of these names are obvious, but not always, because not all object names are the same on all browsers, and not all object properties work on all browsers. This is not as bad as it sounds, because browser developers are making progress in implementing standards from the W3C.

Sometimes people write HTML pages leaving out what seem to be critical elements, such as `HTML`, `HEAD`, and `BODY`. Under DOM rules, the corresponding objects for these missing elements are automatically created because their existence is assumed. Otherwise, there would be no document object from which to refer to all the other objects on the page.

Cross-browser compatibility and the DOM

The DOM as recommended by the W3C is currently only available in fifth-generation browsers. Other browsers support a less complete DOM, which means that not all features of the DOM are available (access to all events, recognition of all user events, and so on).

This complicates your life. You'll need to create your scripts that interact with the DOM very carefully, and employ a lot of error checking, object detection, and browser sniffing to make sure you create stable, dependable scripts that don't crash the user's browser. This is especially true if you are trying to maintain backward compatibility.

See Chapter 48 for a way to look for specific DOM features on a browser, and a method to determine which browser is in use for exclusionary methods of writing DHTML.

In a nutshell, there's really no such thing as cross-browser compatibility. In the next chapter, you'll see a method for detecting which parts of the DOM are supported on a given browser before the code is executed. This is a good way to implement backward compatibility. The other option is to do what accessibility purists really don't like — write your DHTML pages to run only on browsers that are fully compatible with the W3C DOM, and skip execution on all others.

A comprehensive reference guide to DOM objects is available at the MSDN Online Web Workshop site (`http://msdn.microsoft.com/workshop/`). Netscape includes additional information and links to the DOM (`http://developer.netscape.com/tech/dom/`).

The structure of the DOM

Relationships between objects in the DOM are patterned after relationships between family members. An object can have a parent, a child or children, and one or more siblings. Sibling objects share a parent. The Adam and Eve of this family is the *document* object. The document object has no parent, and the list is defined as follows:

```
<OL name="this-list">
<LI name="first-child">
<LI name="second-child">
```

```
    <OL name="third-child">
         <LI name="first-grandchild">
         <LI name="second-grandchild">
    </OL>
  </OL>
```

In the preceding HTML snippet, *this-list* is a child of the document. The list items *first-child*, *second-child*, and *third-child* are all children of *this-list* and siblings to each other. The nested list items *first-grandchild* and *second-grandchild* are children of *third-child*.

All objects within a document are referenced by what type of object they are and what occurrence of this type of object they are, beginning with 0 for the first occurrence. For example, if this-list was the first list on the page, it would be ordered list 0. The second-child would be list item 1 of ordered list 0.

Referring to objects in the DOM is relatively simple. If you have a named object, you can refer to it as itself. Consider the following HTML, which defines the object:

```
<IMG src="images/volcano.gif" name="volcano">
```

In the JavaScript example below, you can easily refer to the image object named above.

```
<SCRIPT language="javascript">
objImg = volcano;
</SCRIPT>
```

If the element definition had omitted the name attribute — for whatever reason — the HTML would look instead, like this:

```
<IMG src="images/volcano.gif">
```

As the page author, you can still refer to the image using a relative method of addressing. The relative method selects the image based on its position within other images in the document. In this example, if the object previously defined is the first instance of an image on the page, then the JavaScript definition would look like this:

```
objImg = document.image[0];
```

In the implementation of the DOM, numbering systems begin at zero, so the first instance of an object is 0, the second is 1, and so on.

In addition to being able to refer to an object by name using the DOM, you can also determine or *set* the value of attributes of the element by getting at an object's *properties*. Every object has properties. With few exceptions, the object's properties

parallel the object's attributes. This even includes, for example, formatting attributes such as the `align` attribute, which is deprecated in most elements (tables being one case where there are no alternatives, because CSS doesn't support table formatting well). Properties are available for deprecated attributes.

Other DOM objects

In addition to querying the value of a property, you can also change the value of the property. This is the power of the DOM. After a page has been loaded, you can change the value of an element's attributes by changing the object's properties. Using the `style` property and subproperties of the `style` property, you can change anything about the look of an element you can set with CSS, the DOM, and the `style` attribute. Style becomes a DOM object:

- `style.color` changes the color of the object.
- `style.textDecoration` changes the text to be underlined, overlined, struck-through (line-through), or to blink.
- `style.fontFamily` changes the font of the text.
- `style.fontSize` changes the size of the text.

As you may recall learning in Part IV of this book, CSS elements have a display attribute. The three common values are `""` (empty value), `"none"` (meaning not visible and taking up no space on the page), and `"hidden"` (meaning not visible, but taking up space on the page anyway). If you want to set an object as invisible and not to take up any space on the page — in other words, to disappear as if it had never been there — after the page has been loaded, usually as the result of an event, you can use the following JavaScript code:

```
document.all.image(0).display = "none";
```

Notice the property name is tacked onto the end of the object name. In this case, the object is being referred to by relationship. From the object name, you can tell this is the first image on the page. The next example refers to the object by name.

```
objImg.display = "none";
```

The object was named *objImg*, but you can't tell what kind of object it is from the JavaScript alone. Whatever kind of object it is, it will be rendered invisible and the rest of the page will be redrawn, so this invisible object doesn't take up any space on the page as a result of this JavaScript.

In addition to showing or hiding objects, you can change almost every property CSS can affect. The fastest way to apply style qualities to an object is to define a style within the `HEAD` element and to apply the entire style by name. In the following

example, you can see the STYLE element in the HEAD, and then the JavaScript you would use to set the style of an object to a different value.

```
<HEAD>
<STYLE>
  beginning {
color: black;
font-family: Helvetica;
font-size: 12pt;
}
alternate {
color: red;
font-family: Helvetica;
font-size: 14pt;
}
</STYLE>
<!-- and later on in the document, in the JavaScript, you could
use the following code -->
document.all.div(4).style = alternate;
<!-- more JavaScript would follow, then the rest of the rest of
the page definition -->
```

Notice the DOM object is assigned directly to the name of the style.

DOM event model

To enable interactive documents, the DOM includes an event model to enable it to respond to a wide range of user actions and changes in page status. In previous implemented versions of the DOM, events have been tied to specific objects. For example, hyperlinks and buttons were the only objects that could detect mouse clicks. You could detect the mouse moving over a hyperlink, but not over a heading. A key part of the W3C DOM is that *all* objects recognize *all* events. This will be a good thing, once it's implemented, enabling you to turn a Web page into a feature-rich graphical user interface.

Another feature of the DOM is called *event bubbling*. When implemented, this feature can pass off event handling for an object to the object's parent. For example, clicking on a form button is initially handled by the event handler for that element. With event bubbling, further handling can then be passed off to the form to which the element belongs. If needed, further handling can then be passed up to the document or other parent structure.

From Here

If you're interested in learning more about JavaScript, go to Chapter 48 for more in-depth information.

Jump to Chapter 49 to experiment with creating cross-browser DHTML.

Summary

The DOM is the greatest thing to hit the Web since forms. It adds a whole new dimension of functionality to Web sites. Together with CSS (especially the capability to place objects in all three dimensions) and JavaScript, the DOM can literally make your page dance. Thanks to the DOM, every element of your page is completely addressable. In addition, the DOM demands that every browser redraw the page dynamically, meaning if you change the definition of an element after the page has been rendered, the browser will redraw the page without having to reload the page from the server.

You can address objects on your page either by name — if they have names and you know them — or by relationship (the child of my parent's third sibling). The exciting thing about the DOM is you take actions on objects by changing their properties; it is this simple. Properties correspond almost identically to attributes in the HTML definition of the object.

Unfortunately, the conclusion to this chapter of the DOM is not as rosy as it should be. Microsoft and Netscape still haven't implemented the DOM the same way. In fact, before a DOM existed, there were just two browser developers trying to offer some level of scripting to their developers. Microsoft goes beyond what the DOM demands while Netscape falls short. Everyone hopes that someday all popular browsers will fully (and identically) support the DOM.